# Importing Isabelle Formal Mathematics into NuPRL

Pavel Naumov*

Department of Computer Science, Cornell University, Ithaca, NY 14853
pavel@cs.cornell.edu

**Abstract.** Isabelle and NuPRL are two theorem proving environments that are written in different dialects of ML using different formula syntaxes and different logical foundations. In spite of this, they have similar sets of basic theories, representing the same mathematical concepts. This paper presents the design of an automated converter from Isabelle into NuPRL that allows sharing formal knowledge between these two provers. Such sharing eliminates the need for re-proving the same results in different systems and opens the door to joint work on large verification projects.

## 1 Introduction

When formal theorem provers first arrived, their compatibility was not regarded as an important issue. Instead, researchers were predominantly looking for better mathematical foundations, better syntax, and better system design. As a result, although many modern proof development environments have unique features that make them in some respects superior to others, they all face the same incompatibility problem.

The temptation to share formal results between different systems grows with the size of formal libraries developed world-wide using different provers. The bigger these libraries become the more obvious it is that translating formal results from one system into another may be less time-consuming than re-proving almost identical theorems from scratch. One of the first works in this area has been done by Howe in [8] and [7]. He developed semantical foundations and semi-automated procedure for importing Mathematics from HOL [6] into NuPRL [4]. Later, in a joint work with Felty [5], he used this translation to conduct proofs in a hybrid HOL-NuPRL system.

This paper presents an attempt to develop a general methodology for converting formal results from Isabelle [12] into NuPRL. Unlike HOL or NuPRL, Isabelle is a "generic" theorem prover, in which different mathematical theories can be formalized by *declaring* appropriate syntax and inference rules in Isabelle meta-logic. These declarations constitute Isabelle *object logic* dedicated to a particular mathematical theory. NuPRL lacks the notion of object logic

* This work was supported by DARPA grant F30602-98-2-0198 "An Open Logical Programming Environment: a Practical Framework for Sharing Formal Models".

because it is based on a pre-specified theory – Constructive Type Theory [9]. Each new mathematical notion should be defined in NuPRL via standard Type Theory constructors. Choosing these definitions is a creative task that can not always be accomplished. Hence, it is impossible to develop a universal converter from Isabelle into NuPRL that will be able to handle all possible object logics. In this paper we focus on development of a general conversion procedure that can be manually adjusted to different object logics. At the present moment the converter works with Isabelle Meta Logic and Isabelle Higher Order Logic. We expect that the same methodology can be applied to the majority of other Isabelle object logics.

The major difference between our work and [7], besides the use of different source systems, is the way the logical soundness of the interpretation is proven. While Howe is using Set Theory semantics for NuPRL, we will give direct syntactical proof. The syntactical approach seems to be easier and more straightforward.

Just like Howe [7], we translate into NuPRL only definitions and statements of theorems. It means that we force NuPRL to *trust* in soundness of Isabelle/HOL proofs. Forcing one prover to rely on the results of another is a debatable approach the legitimacy of which depends on the task to which automated reasoning is applied.

In works, such as [5], which are oriented toward hardware or software verification use of hybrid systems, this approach seems to be perfectly acceptable because the final goal of verification is to find possible bugs. It is probably also admissible to use hybrid systems to look for proofs of new mathematical results. After all, once such a proof is found, it potentially can be checked by a human being or can be repeated in a single system.

On the other hand, hybrid proofs should be less welcomed when a proof development system is used to provide a rigorous formalization of a mathematical theory because in a hybrid proof we have to rely not only on the soundness of both systems, but also on some kind of a meta-argument to claim potential derivability of translated results. This meta-argument normally would be presented as an interpretation of the source system logic in the logic of the target system. For these reasons a converter that translates the entire proofs would be more appealing.

Although Isabelle Meta Logic is a constructive theory, we have decided to convert its theorems into statements in a classical extension of NuPRL Type Theory. In the next section we discuss the rationale behind this decision and define this classical extension. The following section describes embedding of Isabelle syntax into NuPRL and sketches the converter design. The last part of this paper shows logical soundness of the embedding.

## 2 Classical Extension of NuPRL Type Theory

The main reason for using a classical extension of NuPRL is that we want the results, brought over from Isabelle theories to be useful in NuPRL proofs. Thus, ideally, Isabelle type of meta-propositions $o$ should be mapped into NuPRL type of propositions $\mathbb{P}$. Unfortunately, this scheme is hard to implement since NuPRL type $\mathbb{P}$ has infinitely many elements and Isabelle type $o$ is assumed to have only two elements: true and false. It means that, for example, Isabelle meta rule

$$\frac{\begin{array}{cc}[\phi] & [\psi] \\ \psi & \phi\end{array}}{\phi \equiv_{prop} \psi}$$

would not be valid when Isabelle propositions are interpreted as NuPRL propositions and Isabelle equality $\equiv$ is translated as NuPRL equality. A possible way around this problem would be to interpret Isabelle type $o$ as NuPRL quotient type $Q = \mathbb{P}//(x, y.x \Leftrightarrow y)$. Type $Q$ is a factorization of type $\mathbb{P}$ by the equivalence relation "if and only if". From author's experience, dealing with quotient propositions in NuPRL is not an easy task, and converting theorems stated as quotient propositions into standard NuPRL ones is not trivial either. But the most importantly, type $Q$ does not solve our problems. Unlike Isabelle Meta Logic, in NuPRL propositions do not all belong to the same type. They are split into a chain of propositional types of different levels:

$$\mathbb{P}_1 \subseteq \mathbb{P}_2 \subseteq \ldots \subseteq \mathbb{P}_n \subseteq \ldots$$

such that an equality of two propositions of level $n$ is actually an element of type $\mathbb{P}_{n+1}$. It means that type $Q$ in NuPRL is also a sequence of types $Q_n = \mathbb{P}_n//(x, y.x \Leftrightarrow y)$. If we choose some $Q_{n_0}$ to be the interpretation of Isabelle type $o$, then the equality on elements of $o$ would need to be translated as an equality of $Q_{n_0}$ elements which in NuPRL belong to a higher-level type $Q_{n_0+1}$.

Therefore, there is, probably, no reasonably simple modification of NuPRL type $\mathbb{P}_i$ that can be used as an interpretation of Isabelle type $o$.

All mentioned above problems can be avoided by interpreting Isabelle type $o$ as NuPRL boolean type $\mathbb{B}$. Type $\mathbb{B}$ has only two elements and there is boolean equality $=_b$ on type $\mathbb{B}$ such that for any elements $x$ and $y$ of type $\mathbb{B}$, $x =_b y$ is also an element of type $\mathbb{B}$.

Although Isabelle theorem statement will be converted into elements of the NuPRL boolean type, such results still can be easily applied in NuPRL proofs, since NuPRL provides the "assert" function that maps booleans into propositions.

**Boolean Equality** On the other hand, type $\mathbb{B}$ brings problems of its own. In NuPRL not every type has a boolean equality relation. In fact, only decidable types can have boolean equality in NuPRL because boolean equality, just as any other NuPRL function, would be assumed to be computable. We will extend NuPRL Constructive Type Theory by a boolean equality predicate for any type. Such an extension will be non-constructive, or, in other words, classical. Boolean equality is a three-place predicate that takes any type $T$ and two elements $x$ and $y$ of type $T$ as arguments.

There are three inference rules to deal with this function:

$$\frac{H \vdash x_1 = x_2 \in T_1 \qquad H \vdash y_1 = y_2 \in T_1 \qquad H \vdash T_1 = T_2 \in \mathbb{U}}{H \vdash eq(x_1, y_1, T_1) = eq(x_2, y_2, T_2) \in \mathbb{B}}$$

$$\frac{H \vdash eq(x, y, T) = true \in \mathbb{B}}{H \vdash x = y \in T}$$

$$\frac{H \vdash x = y \in T}{H \vdash eq(x, y, T) = true \in \mathbb{B}} \ .$$

We intend to use boolean equality as a translation of Isabelle equality operator. For example, Isabelle axiom $a =_\sigma a$ will be translated into NuPRL as

$$\forall \sigma : \mathbb{S}. \forall a : \sigma. \uparrow eq(a, a, \sigma) \ .$$

**Hilbert $\varepsilon$-operator.** In order to interpret Isabelle Higher Order Logic, one more primitive operator should be added to NuPRL – Hilbert $\varepsilon$-operator. For any non-empty type $T$ and function $b : T \to \mathbb{B}$, operator $\varepsilon t : T.b(t)$ is defined to be equal to any element $t_0$ of type $T$ such that $b(t_0)$ is true. If such element $t_0$ does not exist, then $\varepsilon t : T.b(t)$ returns any element $t$ of type $T$.

Just like boolean equality, the Hilbert operator can be defined in NuPRL for some specific cases, e.g. when type $T$ is finite and proposition $b(t)$ is decidable. But in general it can not be done, since there is no function equivalent to the Hilbert operator in the standard NuPRL model [1]. We will add the Hilbert operator as a new primitive abstraction to NuPRL along with the following two inference rules.

$$\frac{H \vdash T_1 = T_2 \in \mathbb{U}_i \qquad H, t : T \vdash b_1(t) = b_2(t) \in \mathbb{B} \qquad H \vdash t_0 = t_0 \in T_1}{H \vdash \varepsilon t : T_1.b_1(t) = \varepsilon t : T_2.b_2(t) \in T_1}$$

$$\frac{H \vdash b(t_0) = true \in \mathbb{B}}{H \vdash b(\varepsilon t : T.P(t)) = true \in \mathbb{B}} \ .$$

For any boolean function $f : T \to \mathbb{B}$ one can define *boolean existential quantifier*

$$\exists_b t : T.f(t) = f(\varepsilon t : T.f(t))$$

and *boolean universal quantifier*

$$\forall_b t : T.f(t) = \neg_b \exists_b t : T. \neg_b f(t) \ ,$$

where $\neg_b$ is NuPRL boolean negation. We will use these boolean quantifiers to interpret Isabelle quantifiers.

The extension of NuPRL Type Theory by boolean equality and $\varepsilon$-operator and the above five inference rules will be called *Classical NuPRL.* Note that both boolean equality and the Hilbert operator can be defined through an extract of the excluded middle law. Hence, an alternative approach would be to extend NuPRL by excluded middle law and its extract.

# 3 Term Translation

## 3.1 Overview

There are two levels of term representation in Isabelle. Initially each term is written by a user as a string of characters. After this string is parsed by the internal Isabelle parser, it is converted into an element of the ML abstract data type `term`. Such abstract structure contains extra type information about the term, that was re-constructed using the theory signature. For the purpose of our work, this ML data structure is considered to be the Isabelle term, since our translation and the soundness proof will rely on this extra information.

We intend to define a translation of ML structures of type `term` into ML structures, representing NuPRL terms. Although it is possible to translate Isabelle terms directly into NuPRL terms, we will be using intermediate data type *portable term*. The portable term, essentially, is a stripped down version of Isabelle term that does not contain any information that will not be needed for its conversion into a NuPRL term. First, an Isabelle term is being converted into a portable term. Afterwards, portable term is translated into a NuPRL term. Use of the intermediate structure, portable term, hence, leads to a cleaner converter design, where two independent translation stages are performed separately.

In addition, the portable term format is used as a bridge between two different dialects of ML in which Isabelle and NuPRL are written. Isabelle is using Standard ML of New Jersey [10] and NuPRL is based on Classic ML. Hence, at some point during translation a data structure should be transmitted from one ML environment into another. In our implementation, a term is passed from one system into another in the portable format.

There are two different definitions of the portable term data structure. One is done in SML and another in Classic ML. Both of them represent the same mathematical abstraction. Technically, a conversion from SML to a Classic ML is done by SML function `pterm2string` that produces Classic ML description of any SML portable term. The output of this function can be dumped into a file which will construct Classic ML portable term after being loaded into a Classic ML environment (See Figure 1).

## 3.2 Type Term

Isabelle distinguishes two kinds of terms: type terms and regular terms. Type terms represent Isabelle types and regular terms encode elements of these types. This distinction is preserved in the portable format. The Isabelle type term is defined as

```
datatype typ = Type  of string * typ list
               TFree of string * sort
               TVar of indexname * sort

type indexname = string * int
```

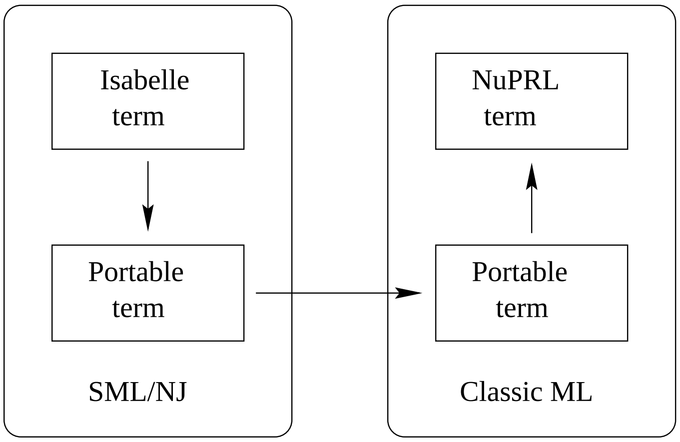


**Fig. 1.** Translation of Isabelle terms into NuPRL

The notion of type sorts and a related idea of type classes are Isabelle-specific and probably can not be translated into NuPRL Type Theory in a natural way. We will interpret all Isabelle types as non-empty elements of the NuPRL first-level type universe $\mathbb{U}_1$. Hence, we do not need the information about `sorts` in portable terms. Also, free variables and scheme variables differ only in the way they are treated by Isabelle unification procedure. For the purposes of the converter they can be considered to be the same. Taking this into account, portable type terms are defined in SML as:

```
datatype ptyp = PType of string * ptyp list | PTVar of string
```

An easily definable SML function `typ2ptyp` converts Isabelle terms into the portable format.

### 3.3 Regular Term

Elements of Isabelle types are encoded into SML as data structures of the type `term`

```
datatype term = Const of string * typ
              | Free  of string * typ
              | Var   of indexname * typ
              | Bound of int
              | Abs   of string * typ * term
              | op    of term * term
```

Accordingly, we define the portable term type `pterm` as

```
datatype pterm = PConst of string * ptyp
               | PVar   of string * ptyp
```

```
                  | PBound of int
                  | PAbs   of string * ptyp * pterm
                  | Pop    of pterm * pterm
```

We have also defined an SML function `term2pterm` that converts Isabelle terms into portable format.

### 3.4 Portable Terms in Classic ML

The same data types `ptyp` and `pterm` can be specified in Classic ML using a slightly different syntax.

```
absrectype ptyp = string # (ptyp list) + string
  with PType(s,l)    = abs_ptyp(inl (s,l))
  and  PTVar(s)      = abs_ptyp(inr s)

absrectype pterm = string # ptyp + string # ptyp + int +
                   string # ptyp # pterm + pterm # pterm
  with PConst(s,t)    = abs_pterm(inl (s,t))
  and  PVar(s,t)      = abs_pterm(inr (inl (s,t)))
  and  PBound(n)      = abs_pterm(inr (inr (inl n)))
  and  PAbs(s,t,m)    = abs_pterm(inr (inr (inr (inl (s,(t,m))))))
  and  Pop(t1,t2)     = abs_pterm(inr (inr (inr (inr (t1,t2)))))
```

### 3.5 Importing Portable Terms into NuPRL

Importing portable terms into NuPRL consists in the translation of terms from portable format into NuPRL term syntax. During this translation both type terms and regular terms are mapped into NuPRL terms because NuPRL Type Theory does not distinguish between these two kinds of expressions.

A Classic ML recursive function `ptyp2term` converts portable type terms into NuPRL terms. It maps portable type constants ``` ``bool'' ```, ``` ``nat'' ```, ``` ``prop'' ```, and ``` ``fun'' ``` into NuPRL types $\mathbb{B}$, $\mathbb{N}$, $\mathbb{B}$, and $\rightarrow$. Portable type variables are being translated to NuPRL variables with the same name.

Classic ML recursive function `pterm2term` converts regular portable term $t$ into a NuPRL term. Because Isabelle is using de Bruijn [3] indices to represent bound variables and NuPRL employs more conventional notations, the definition of `pterm2term` function is not straightforward. Namely, it should keep track of bound variable names, such as `s`, appearing in the lambda constructor `PAbs(s, t, m)`. The primitive portable terms, built with constructor `PConst(s,t)` are converted into NuPRL using a pre-specified table of NuPRL abstractions, corresponding to Isabelle abstractions. For example, portable term constant

```
PConst("op +", PType("fun",[PType("nat",[]);
                            PType("fun",[PType("nat",[]);
                                         PType("nat",[]);
                                        ]);
```

```
                                   ]
                          )
          )
```

is converted into NuPRL term $\lambda n, m.\ n + m$.

### 3.6 Theorem Translation

Unlike Isabelle theorems, NuPRL theorems can not contain free variables. Hence, before any portable term can be stated as a NuPRL theorem, all free variables in this term should be bound. We bind all type variables by a universal quantifier over the universe of non-empty first level types $\mathbb{S} = \{T : \mathbb{U}_1 | T\}$. To understand the set condition in the above formula it is important to remember that NuPRL is using types-as-propositions doctrine. Accordingly, any type $T$ is a proposition. Informally, this proposition states that type $T$ is not empty. Free variables ranging over types are bound by universal quantifiers over corresponding types. The type information can be extracted from portable terms since any free variable in a portable term has form `PVar(s,t)` where `s` is the variable name and `t` is the portable term, representing its type.

It is also important to note that we intend to interpret Isabelle meta-logical two-element type *prop* as NuPRL type $\mathbb{B}$. Hence, the translation of an Isabelle theorem will have boolean type in NuPRL. In order to state them as NuPRL theorems we need to apply the NuPRL operator "assert" that converts booleans into propositions. This operator is denoted in NuPRL by the arrow $\uparrow$.

**Example.** Isabelle theorem $a \Rightarrow a$ will be translated into NuPRL as

$$\forall a : \mathbb{B}.\ \uparrow (a \rightarrow_b a)\ \ ,$$

where $\rightarrow_b$ is NuPRL boolean implication.

### 3.7 Theory Translation

The conversion of mathematical results from Isabelle into NuPRL is done on a theory-per-theory basis. The SML function `export` can be applied to any Isabelle theory. The side effect of the evaluation of this function is the creation of Classic ML file with description of a *portable theory*. The datatype *portable theory* contains the theory name and a list of theorems of this theory in the portable format. Once this file is loaded into Classic ML environment, the function `import` can be applied to portable theory data structure to create corresponding NuPRL theory. The name of this NuPRL theory will be taken from the first field of the portable theory data structure and the location of the theory in NuPRL library should be specified as an argument of the `import` function.

One of the problems appearing during theory translation is associated with name-space handling. There is no simple solution to this problem since the current NuPRL library has a plain name-space in which no two objects can have the same name. To provide a minimal assurance that name collision between imported theorems and existing NuPRL objects will be avoided, we have decided to prefix Isabelle theorem names with the name of the Isabelle theory from which it is taken. For example, Isabelle theorem `Suc_diff_Suc` from theory `Arith` will be called `Arith_Suc_diff_Suc` after it is imported into NuPRL. The other problem, that was discovered soon after the first attempt of translation is that some characters used in Isabelle names are prohibited in NuPRL object name syntax. Among them are space, prime, and period. As a temporary solution, we have decided to substitute "_prime_" for prime, and "_" for the rest of characters in question. Obviously, it creates a potential problem. A much better solution would be to remove unnecessary restrictions from NuPRL syntax.

## 4 Logical Foundation

In the previous section we have described a translation of Isabelle into NuPRL syntax and sketched the implementation of this procedure by two ML functions: `export` and `import`. In this section we will show that this translation provides an interpretation of Isabelle Meta Logic and Isabelle Higher Order Logic in the classical extension of NuPRL Type Theory.

### 4.1 Pure Isabelle

Isabelle meta-logic, also known as *Pure Isabelle*, is an intuitionistic higher-order Type Theory. So is NuPRL. The major difference between the two of them is that Pure Isabelle is a "logical framework" designed to reason about provability in other theories using internal types as a tool while NuPRL is oriented toward the reasoning about internal types themselves. Although important in practice, this difference is not significant from a logical point of view. In fact, as demonstrated in Constable and Howe [2], NuPRL also can be used for reasoning about other theories.

Isabelle meta-logic is axiomatized in [11] as a natural deduction system. Since we intend to define interpretation of Pure Isabelle in Classical NuPRL, it will be more convenient to deal with Gentzen-style axiomatization from Figure 2. The equivalence of two formalizations can be established by a standard Proof Theory technique.

Our next step is to prove that the translation of each Pure Isabelle theorem is derivable in NuPRL by showing that inference rules from Figure 2 are translated into derivable inference rules in NuPRL. But before doing that we need to specify how Isabelle sequents are translated into NuPRL.

Sequent translation is very similar to theorem translation: the assertion operator is applied to all hypotheses and the conclusion. In addition, free variable declarations should be added to the beginning of the hypothesis list. For instance, sequent $\Gamma \vdash a =_\sigma a$ is translated into NuPRL, assuming that $a$ and $\sigma$ are free variables, as

$$\sigma : \mathbb{S}, a : \sigma, \tau(\Gamma) \vdash \uparrow eq(a, a, \sigma) \ \ ,$$

1. $\dfrac{\Gamma,\phi \vdash \psi}{\Gamma \vdash \phi \Rightarrow \psi}$

2. $\dfrac{\Gamma \vdash \phi \Rightarrow \psi \qquad \Gamma \vdash \phi}{\Gamma \vdash \psi}$

3. $\dfrac{\Gamma \vdash \phi}{\Gamma \vdash \bigwedge_\sigma x.\phi}$

4. $\dfrac{\Gamma \vdash \bigwedge_\sigma x.\phi}{\Gamma \vdash \phi[b/x]}$

5. $\dfrac{}{\Gamma \vdash a \equiv_\sigma a}$

6. $\dfrac{\Gamma \vdash a \equiv_\sigma b}{\Gamma \vdash b \equiv_\sigma a}$

7. $\dfrac{\Gamma \vdash a \equiv_\sigma b \qquad \Gamma \vdash b \equiv_\sigma c}{\Gamma \vdash a \equiv_\sigma c}$

8. $\dfrac{}{\Gamma \vdash (\lambda x.a) \equiv_\sigma (\lambda y.a[y/x])}$

9. $\dfrac{}{\Gamma \vdash ((\lambda x.a)b) \equiv_\sigma a[b/x]}$

10. $\dfrac{\Gamma \vdash f(x) \equiv_\sigma g(x)}{\Gamma \vdash f \equiv_{\rho \to \sigma} g}$

11. $\dfrac{\Gamma \vdash a \equiv_\sigma b}{\Gamma \vdash (\lambda x.a) \equiv_{\rho \to \sigma} (\lambda x.b)}$

12. $\dfrac{\Gamma \vdash f \equiv_{\rho \to \sigma} g \qquad \Gamma \vdash a \equiv_\rho b}{\Gamma \vdash f(a) \equiv_\sigma g(b)}$

13. $\dfrac{\Gamma,\phi \vdash \psi \qquad \Gamma \psi \vdash \phi}{\Gamma \vdash \phi \equiv_{prop} \psi}$

14. $\dfrac{\Gamma \vdash \phi \equiv_{prop} \psi \qquad \Gamma \vdash \phi}{\Gamma \vdash \psi}$

**Fig. 2.** Gentzen-style axiomatization for Pure Isabelle

where $\tau(\Gamma)$ is the translation of the hypothesis list $\Gamma$.

Of course, the same inference rule would be invalid in NuPRL if $a$ is a non-well-typed expression such as $true + 1$. The problem comes from the fact that in Isabelle only well-typed terms, called *certified terms* ([12], p. 121), are allowed in proofs but in NuPRL any term can be used in inference rules.

Fortunately, to prove the derivability of Pure Isabelle theorems in Classical NuPRL, we only need to guarantee that every *instance* of Isabelle rules is sound in NuPRL. In other words, Isabelle rules are derivable in NuPRL if we restrict them to translations of corresponding Isabelle sequents.

**Theorem 1.** *Translation of any instance of Pure Isabelle inference rules from Figure 2 is derivable in Classic NuPRL.*

*Proof.* The rules from Figure 2 can be divided into two classes. Rules 1, 2, 6, 7, 12, 13, and 14 correspond to appropriate NuPRL rules, and derivability proofs in these cases are straightforward. NuPRL counterparts of the rules 3, 4, 5, 8, 9, 10, and 11 require additional well-formness subgoals that are missing in Isabelle rules. At the same time, if we restrict NuPRL formulas to the translations of certified Isabelle terms, appropriate well-formness follows from the certification condition. Here we consider two typical cases: Rule 1 from the first class and Rule 3 from the second class. The translations of the other rules could be verified similarly.

Translation of Rule 1 into NuPRL can be written as:

$$\frac{map(decl,\Gamma), decl(\phi), decl(\psi), map(\uparrow \circ \tau, \Gamma), \uparrow \tau(\phi) \vdash \uparrow \tau(\psi)}{map(decl,\Gamma), decl(\phi), decl(\psi), map(\uparrow \circ \tau, \Gamma) \vdash \uparrow \tau(\phi \Rightarrow \psi)},$$

where $decl(\alpha)$ is the list of free variable declarations extracted from an Isabelle formula $\alpha$, $map(f, l)$ is the element-wise application of function $f$ to list $l$, $\circ$ is the function composition operator, and $\tau$ is the defined above translation of

Isabelle terms into NuPRL. The validity of the above rule follows from the fact that

$$\tau(\phi \Rightarrow \psi) = \tau(\phi) \rightarrow_b \tau(\psi) \ ,$$

where $\rightarrow_b$ is NuPRL boolean implication.

As has been mentioned above, the verification of Rule 3 significantly relies on the use of certified terms in Isabelle. Since term $\bigwedge_\sigma x.\phi$ is certified in Isabelle, type expression $\sigma$ should be well-formed and variable $x$ should have type $\sigma$ in this term. There are two possible cases:

*Case 1.* Free variable $x$ does not occur in the formula $\phi$. This case is trivial.

*Case 2.* Free variable $x$ does occur in the formula $\phi$. It means that

$$decl(\phi) = d_1, x : \tau(\sigma), d_2$$

for some declaration lists $d_1$ and $d_2$. Hence, the translation of Rule 3 to NuPRL looks like

$$\frac{map(decl, \Gamma), d_1, x : \tau(\sigma), d_2, map(\uparrow \circ\tau, \Gamma) \vdash\uparrow \tau(\psi)}{map(decl, \Gamma), decl(\phi), d_1, d_2, map(\uparrow \circ\tau, \Gamma) \vdash\uparrow \forall_b x : \tau(\sigma).\tau(\phi)}$$

which is a valid NuPRL rule. □

**Corollary 1.** *Translations of Pure Isabelle theorems are provable in Classic NuPRL.*

### 4.2 Isabelle Higher Order Logic

Higher Order Logic (HOL) is probably the most widely used Isabelle object theory. It is defined in [12] as an extension of Pure Isabelle by a new type *bool*, and operators Trueprop, $=$, $\rightarrow$, and @ on this type. Operators have types $bool \Rightarrow prop$, $\alpha \Rightarrow \alpha \Rightarrow \alpha$, $bool \Rightarrow bool \Rightarrow bool$, and $(\alpha \Rightarrow bool) \Rightarrow \alpha$ correspondingly. In addition to Pure Isabelle inference rules, HOL contains the axioms, stated on Figure 3.

We interpret HOL boolean type *bool* as NuPRL boolean type $\mathbb{B}$, *Trueprop* as the identity function, HOL implication $\rightarrow$ as NuPRL boolean implication $\rightarrow_b$, and HOL choice operator @ as NuPRL $\varepsilon$-operator. This makes axioms from Figure 3 obviously valid.

**Theorem 2.** *Translations of HOL Isabelle theorems are provable in Classic NuPRL.*

## 5 Conclusions

The paper presented an automated tool for sharing mathematics between Isabelle and NuPRL. We have used this converter to bring basic Isabelle Higher Order Logic theories into NuPRL environment. The same procedure could be potentially applied to other Isabelle object logics.

$$\mathsf{Trueprop}(t = t)$$
$$\mathsf{Trueprop}(s = t) \Rightarrow \mathsf{Trueprop}(P(s)) \Rightarrow \mathsf{Trueprop}(P(t))$$
$$\bigwedge_\sigma x.\mathsf{Trueprop}(f(x) = g(x)) \Rightarrow \mathsf{Trueprop}(\lambda x.f(x) = \lambda x.g(x))$$
$$\mathsf{Trueprop}(P) \Rightarrow \mathsf{Trueprop}(Q)) \Rightarrow \mathsf{Trueprop}(P \to Q)$$
$$\mathsf{Trueprop}(P \to Q) \Rightarrow \mathsf{Trueprop}(P) \Rightarrow \mathsf{Trueprop}(Q)$$
$$\mathsf{Trueprop}((P \to Q) \to (Q \to P) \to (P = Q))$$
$$\mathsf{Trueprop}(P(x)) \Rightarrow \mathsf{Trueprop}(P(@x.P(x)))$$
$$\mathsf{Trueprop}((P = \mathsf{True})|(P = \mathsf{False}))$$

where

$$\mathsf{True} = (\lambda x.x = \lambda x.x)$$
$$\forall x.P(x) = (\lambda x.P(x) = \lambda x.\mathsf{True})$$
$$\mathsf{False} = (\forall P.P)$$
$$P|Q = \forall R.((P \to R) \to (Q \to R) \to R)$$

**Fig. 3.** Isabelle HOL axiomatization

When using converted results, NuPRL is not only forced to rely on the soundness of an Isabelle logic and its particular implementation in the Isabelle theorem prover. It also has to rely on the soundness of the converter and underlying interpretation. Although a proof of the interpretation soundness is sketched in the previous section, many technical details are skipped. The author is currently finishing the formalization of this proof in NuPRL. Namely, we have defined a deep embedding of Isabelle syntax into NuPRL, specified translation as an internal NuPRL function, and proved its soundness formally.

## 6 Acknowledgements

The author would like to thank Robert L. Constable for numerous discussions and his support of this project, Doug J. Howe for the time and effort of explaining to me the technical details of his work, and Evan Moran for useful comments on the earlier draft of this manuscript.

## References

1. S. Allen. A non-type theoretic definition of Martin-Löf's types. In *Proceedings of the Second Annual Symposium on Logic in Computer Science*, pages 215–221, 1987.
2. R.L. Constable and D.J. Howe. Nuprl as a general logic. In P. Odifreddi, editor, *Logic and Computer Science*, pages 77–89. Academic Press, London, 1990.
3. N.G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to Church-Rosser theorem. *Indag. Math.*, 34:381–392, 1972.

4. R.L. Constable et al. *Implementing Mathematics with Nuprl Proof Development System.* Prentice Hall, 1986.
5. A.P. Felty and D.J. Howe. Hybrid interactive theorem proving using Nuprl and HOL. In W. McCune, editor, *Automated Deduction – CADE-14*, volume 1249 of *Lecture Notes in Artificial Intelligence*, pages 351–365, Berlin, 1997. Springer-Verlag.
6. M.J.C. Gordon and T.F. Melham. *Introduction to HOL – A Theorem Proving Environment for Higher Order Logic.* Cambridge University Press, 1993.
7. D.J. Howe. Importing mathematics from HOL into Nuprl. In J. von Wright, J. Grundy, and J. Harrison, editors, *Theorem Proving in Higher Order Logics*, volume 1125 of *Lecture Notes in Computer Science*, pages 267–282, Berlin, 1996. Springer-Verlag.
8. D.J Howe. Semantics foundation for embedding HOL in Nuprl. In M. Wirsing and A. Nivat, editors, *Algebraic Methodology and Software Technology*, volume 1101 of *Lecture Notes in Computer Science*, pages 85–101, Berlin, 1996. Springer-Verlag.
9. P. Martin-Löf. *Intuitionistic Type Theory.* BIBLIOPOLIS, Napoli, 1984.
10. R. Milner, M. Tofte, R.M. Harper, and D.B. MacQueen. *The Definition of Standard ML (Revised).* MIT Press, Cambridge, 1997.
11. L.C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989.
12. L.C. Paulson. *Isabelle – A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1994.
13. A.S. Troelstra and H. Schwichtenberg. *Basic Proof Theory.* Cambridge University Press, 1996.